УДК 681.03

А.Е. Дорошенко, Г.Е. Цейтлин, В.А. Иовчев

ВЫСОКОУРОВНЕВЫЕ СРЕДСТВА АВТОМАТИЗАЦИИ ПРОЕКТИРОВАНИЯ ПАРАЛЛЕЛЬНЫХ АЛГОРИТМОВ

В работе рассматриваются вопросы создания инструментария, ориентированного на разработку параллельных алгоритмов и программ. Предусматривается реализация системы в распределенной среде, предоставление ей Web-интерфейса и специализация системы в конкретных предметных областях, в которых алгебро-алгоритмические спецификации и превращения алгоритмов объединены с мощными средствами автоматизации и синтеза программ на основе Web 2.0.

Введение

Появление новых архитектур микропроцессоров (таких как мультиядерных), а также новых классов и вычислительных систем (таких как кластеры, P2P, Грид-системы) обнаружило новые возможности для параллельных вычислений, и соответственно, привело к потребности создания специальных инструментов для разработки (параллельного) программного обеспечения таких систем. Главным источником увеличения производительности программ для таких систем, как и раньше, остается эффективное использование возможностей распараллеливания операций. А именно операций разных уровней выполнения вычислений в программах. Но проблема заключается не только в потребности создания инструментов, которые конструируют новые классы параллельных программ, но и в обеспечении возможностей гибкого проектирования таких программ для их быстрой реинженерии на другие платформы.

Индустриальный опыт параллельного программирования ориентирует конечного пользователя на использование низкоуровневых средств на уровне языка программирования. Низкоуровневые инструменты обычно эффективны, однако их широкое применение вряд ли возможно именно ввиду их низкоуровневой природы, поскольку они требуют много ручной работы и достаточно высокой квалификации программиста. Фокусируя внимание на конечном этапе разработки – кодировке программы на языке программирования – эти средства лишь идентифицируют проблемы и после анализа, и тестирования программы требуют повторного ручного изменения кода начальной программы для того, чтобы проанализировать его опять для нахождения новых ошибок и дефектов.

Главные усилия исследователей в решении проблем мультипоточного программирования, в настоящее время сконцентрированы на изобретении некоторых “простых и практических” общих механизмов, которые позволили бы решить вопрос мультиядерного выполнения программ “почти” автоматически. Несмотря на рациональность этих подходов и примеры их успешного применения, они не стали панацеей для универсального решения названных проблем в силу того, что разработка эффективных программ для мультиядерной архитектуры оказалась очень сложной и масштабной научно-технической проблемой. Очевидно, их успешное решение возможно на пути постепенного углубления и охвата этапов жизненного цикла разработки программы с применением средств автоматизированного проектирования и программирования, от написания первичных спецификаций к генерации выполняемого кода программы. Такой же, основанный на привлечении широких инженерных знаний о проектируемой распределенной системе, путь кажется перспективным и для автоматизации разработки параллельных кластерных систем и Грид-программ.

В Институте программных систем НАН Украины на протяжении длительного периода развивается теория, методология и инструментарий (система ИПС) для автоматизированного проектирования параллельных программ, основанные на

средствах высокоуровневой алгебро-алгоритмической формализации и автоматизации превращений программ [1–5]. Разработана экспериментальная инструментальная система для конструирования и оптимизации параллельных программ [6–8]. Другая система символьных вычислений TermWare, создана на основе парадигмы переписывающих правил, дополняет возможности системы ИПС и используется для автоматизации инженерии программного кода параллельных программ [9–10]. Разрабатываемый ныне инструментарий системы ИПС предназначен для проектирования параллельных программ для современных сред выполнения, в которых алгебро-алгоритмические спецификации и превращения алгоритмов объединены с мощными средствами автоматизации и синтеза программ на основе Web 2.0. Отметим, что под современными средами выполнения понимаются мультиядерные, мультипотоковые, кластер, Грид-системы, облачные системы и т.д.

Изложение материала работы подчинено следующей структуре. В разделе 1 рассматривается использование равносильных форм проектирования алгоритмов и программ. В разделе 2 изложена сущность аппарата ГСП, сочетающего понятия САА-М с проектированием алгоритмов в терминах грамматического вывода. Раздел 3 посвящен проектированию и синтезу параллельных объектно-ориентированных программ. В разделе 4 приведена архитектура онлайнового инструментального средства проектирования алгоритмов и программ. Раздел 5 завершает работу описанием результатов и перспектив.

1. Использование равносильных форм проектирования алгоритмов и программ

Примером открытых систем схемного проектирования классов алгоритмов и программ является синтезатор МУЛЬТИПРОЦЕССИСТ [11]. Данная система, используя алгоритмические проекты, оформленные на языке САА-схем, генерирует тексты программ на целевых языках программирования (Ассемблер, Си, Паскаль и др.). САА-схемы представляют собой естественно-лингвистические проекты алгоритмов, в основу которых положен аппарат систем алгоритмических алгебр (САА) Глушкова. В Украине первая алгебра программ была построена В.М. Глушковым в рамках работ по системам алгоритмических алгебр [1] в середине 60-х годов, за несколько лет до того, как задача построения подобной алгебры была предложена Э. Дейкстрой. Более того, еще в 1959 году в [12] Л.А. Калужнин предложил алгебру граф-схем как инструмент для математизации программирования, которая и стала отправной точкой для создания САА. В 1982 году была создана система синтеза программ МУЛЬТИПРОЦЕССИСТ [11, 13], в качестве входного языка была использована параллельная модификация САА– САА-М.

Рассматриваемый инструментарий (в отличие от системы МУЛЬТИПРОЦЕССИСТ) состоит в интеграции всех трех представлений алгоритма [2]: аналитического (формула в избранной алгебре), естественно-лингвистического (САА-схемы) и графового (граф-схемы Калужнина) при его конструировании.

Аналитическое представление, как известно, базируется на алгебрах и является компактной записью алгоритма, направленной на его дальнейшее преобразование (минимизация, оптимизация по разным критериям) на базе аппарата соотношений и тождеств, развитых в алгебрах.

Естественно-лингвистическое представление (в виде текста), также базируется на аппарате алгебр, в процессе модификации с помощью метаправил декомпозируется на инвариантную часть (неинтерпретированную схему), которая представляет собою верхний уровень проекта и собственно интерпретации (нижний уровень), зависящие от избранной предметной области. После свертки инвариантная часть может быть снова проинтерпретирована, но уже с помощью других средств нижнего уровня.

Графовое представление, главное преимущество которого – наглядность, также опирается на аппарат алгебр и ориентировано на визуализацию конструируемого алгоритма. Основой для построе-

ния этого представления (для аналитического также) может быть выбрана инвариантная часть САА-схемы с дальнейшей интерпретацией в соответствии с избранным нижним уровнем.

Пример 1. Проблема синтаксического анализа состоит в установлении правильности обрабатываемых символьных цепочек - их принадлежность к соответствующему входному языку (задача контроля) и определение синтаксической структуры анализируемых цепочек (задача анализа). Для решения проблемы синтаксического анализа представлен известный метод Кока – Янгера – Касами (Cocke – Younger – Kasami), использующий контекстно-свободные грамматики, представленные в бинарной форме, для описания формальных языков. В качестве результата получаем ответ на вопрос, выводима или нет анализируемая терминальная цепочка символов в данной грамматике [14–16]. Алгоритм принимает на вход грамматику в бинарной форме. Заметим, что исходная грамматика не содержит бесполезных символов, это является обычным требованием к грамматикам, попадающим на вход алгоритмов синтаксического анализа.

Последовательный CYK алгоритм затрачивает на разбор строки длинной n время $O(n^{2,83})$ и память емкостью n^2 на модели RAM [5]. Метод гарантирует выполнение той же работы для произвольной грамматики за время, пропорциональное кубу длины входной цепочки и, таким образом, является одним из методов «динамического программирования», где вычисление последующих шагов опирается на значение, полученные на предыдущих. CYK алгоритм строит множества нетерминалов, соответствующих некоторым участкам входной строки и, в результате, отвечает на вопрос, выводима ли данная строка в данной грамматике или нет.

Метод работает следующим образом. Пусть $G(N,T,P,@)$ – КС-грамматика без ***e***-правил в бинарной форме. Простое обобщение алгоритма работает и для грамматик, не находящихся в этой форме.

Пусть $w = a_1 \ldots a_n$ – входная строка, которую нужно разобрать согласно грамматике G. Предполагается, что a_i *принадлежит* T для $1 <= i <= n$. Суть алгоритма состоит в построении треугольной таблицы разбора, элементы которой обозначим R_{ij}, где $1 <= j <= n - i + 1$. Значениями элементов R_{ij} будут подмножества множества N. Нетерминальный символ A будет принадлежать R_{ij} в случае, когда $A \rightarrow^{+} a_i a_{i+1} \ldots a_{i+j-1}$, т.е. когда из A выводятся j входных символов, начиная с позиции i. В частности, входная строка w принадлежит языку $L(G)$ только в том случае, если *@-аксиома принадлежит* R_{1n}.

Таким образом, чтобы выяснить принадлежит ли строка w языку $L(G)$, CYK-алгоритм вычисляет таблицу разбора для w и анализирует, принадлежит ли *@-аксиома* ее элементу R_{1n}.

При параллельном вычислении таблицы эффективно используется стратегия двусторонней символьной мультиобработки.

Естественно-лингвистическое представление последовательного CYK алгоритма имеет вид:

```
СХЕМА CYK====
"САА схема последовательного CYK алгоритма"
КОНЕЦ КОММЕНТАРИЯ
"CYK"

===="СТАРТ"
ЗАТЕМ
ЦИКЛ
ПОКА НЕ ‘Обработаны все диагонали (d = 1, …, n)’
ВЫЧИСЛИТЬ_ДИАГОНАЛЬ(d)
КОНЕЦ ЦИКЛА
ЗАТЕМ
ЕСЛИ ‘Аксиома принадлежит элементу R[n, 1]’
ТОГДА СООБЩЕНИЕ «входная цепочка правильна»
ИНАЧЕ СООБЩЕНИЕ «входная цепочка не правильна»
КОНЕЦ ЕСЛИ
ЗАТЕМ
"ФИН"

"ВЫЧИСЛИТЬ_ДИАГОНАЛЬ (d)"
===="СТАРТ"
ЦИКЛ
```

ПОКА НЕ ‘Обработаны все элементы диагонали (k = 1, …, n– d + 1)’
ВЫЧИСЛИТЬ_ЭЛЕМЕНТ(d, k)
КОНЕЦ ЦИКЛА
ЗАТЕМ
"ФИН"

"ВЫЧИСЛИТЬ_ЭЛЕМЕНТ (i, j)"
====="СТАРТ"
ЗАТЕМ
“Элемент R[i, j] = 0”
ЗАТЕМ
ЕСЛИ ‘Вычисление первой строки (i = 1)’
ТОГДА
ЦИКЛ
ПОКА ‘Существует правило A > aj’
“Поместить A в элемент R[i, j]”
КОНЕЦ ЦИКЛА
ИНАЧЕ “Установить указатель k на первую диагональ (k = 1)”
ЗАТЕМ
ЦИКЛ
ПОКА НЕ ‘Указатель k достиг i-ой диагонали’
ЦИКЛ ‘Для каждого нетерминала B из элемента R[k, j]’
ЦИКЛ ‘Для каждого нетерминала C из элемента R[i - k, j + k]’
ЕСЛИ ‘Если существует правило A > BC’
ТОГДА “Поместить A в элемент R[i, j]”
КОНЕЦ ЕСЛИ
КОНЕЦ ЦИКЛА
КОНЕЦ ЦИКЛА
КОНЕЦ ЦИКЛА
КОНЕЦ ЕСЛИ
ЗАТЕМ
"ФИН"
КОНЕЦ СХЕМЫ

Аналитическое представление алгоритма имеет следующий вид:

CYK алгоритм = {[u1] B} * ([u2] A1, A2)

u1 – условие: Обработаны все диагонали;

B – оператор: ВЫЧИСЛИТЬ_ДИАГОНАЛЬ;

u2 – условие: Аксиома принадлежит элементу R[n, 1];

A1 – СООБЩЕНИЕ «входная цепочка правильна»;

A2 – оператор: СООБЩЕНИЕ «входная цепочка не правильна»;

B = {[v1] C}

v1 – условие: Обработаны все элементы диагонали;

C – оператор: ВЫЧИСЛИТЬ_ЭЛЕМЕНТ;

C = C1*([w1] {[w2] C2}, C3 * {[w3] {[w4] {[w5] ([w6] C4, E)}}})

C1 – оператор: Элемент R[i, j] = 0;

w1 – условие: Вычисление первой строки;

w2 – условие: Существует правило A > aj;

C2 – оператор: Поместить A в элемент R[i, j];

C3 – оператор: Установить указатель k на первую диагональ;

w3 – условие: Указатель k достиг i-ой диагонали;

w4 – условие: Для каждого нетерминала B из элемента R[k, j];

w5 – условие: Для каждого нетерминала C из элемента R[i - k, j + k];

w6 – условие: Если существует правило A > BC’;

C4 – оператор: Поместить A в элемент R[i, j]”;

E – оператор: Пустой.

Граф-схема последовательного CYK алгоритма показана на рис. 1, и 2 ВЫЧИСЛИТЬ ДИАГОНАЛЬ (d), на рис. 3 ВЫЧИСЛИТЬ ЭЛЕМЕНТ (i, j):

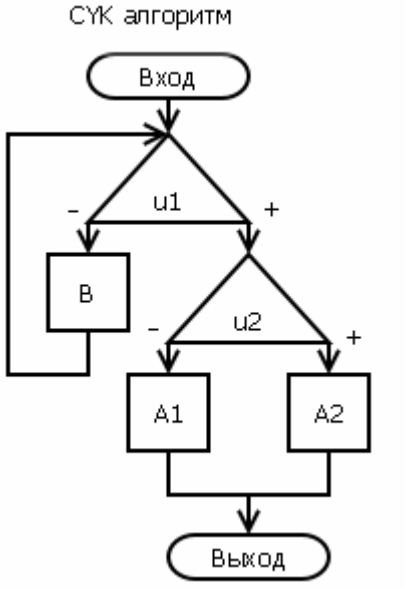

Рис. 1

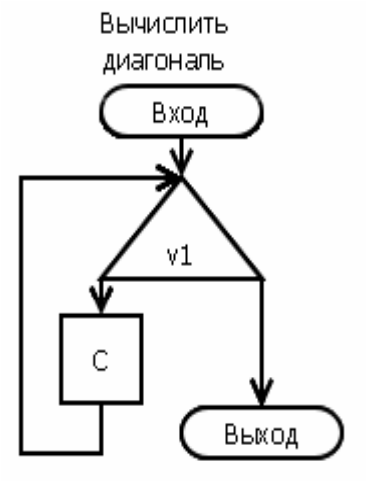

Рис. 2

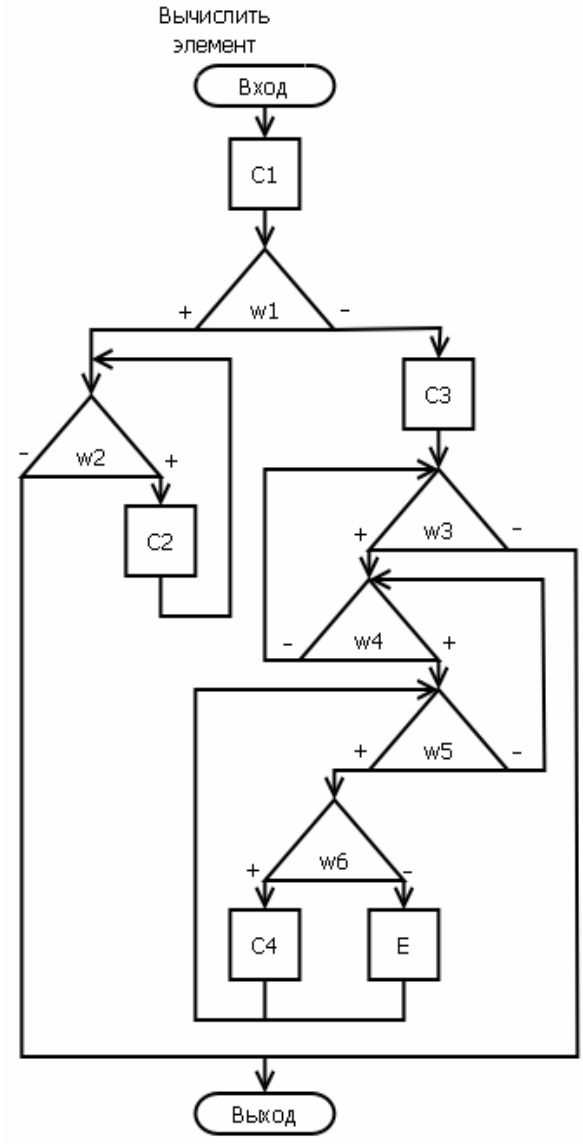

Рис. 3

СХЕМА CYK====
"САА схема CYK алгоритма синхронного вычисления элементов диагоналей"
КОНЕЦ КОММЕНТАРИЯ
"CYK"
====“СТАРТ"
ЗАТЕМ
ЦИКЛ
ПОКА НЕ ’Обработаны все диагонали’
“Распределить количество элементов, обрабатываемой диагонали, по n потокам”
ЗАПУСТИТЬ ВЕТВЬ(1, 2, ..., n)
ОЖИДАТЬ ‘Все ветви обработали выделенные им элементы’
КОНЕЦ ЦИКЛА
ЗАТЕМ
“Проверить наличие аксиомы в элементе R[n, 1]”
ЗАТЕМ
"ФИН"

"ЗАПУСТИТЬ ВЕТВЬ(i)"
====“СТАРТ"
ЗАТЕМ
ЦИКЛ
ПОКА НЕ ‘Вычислены все элементы, выделенные для обработки’
“Вычислить элемент”
КОНЕЦ ЦИКЛА
"ФИН"
КОНЕЦ СХЕМЫ

В приведенной схеме выполняется равномерное распределение элементов, обрабатываемой диагонали, по всем потокам. Распределение предполагается проводить по следующей формуле:

array[index] = count DIV proc;

если index <= (count MOD proc), тогда array[index] = array[index] + 1;

где array – массив, в котором хранятся количества обрабатываемых элементов, для каждого потока; index – порядковый номер потока; count – количество элементов на обрабатываемой диагонали; proc – количество потоков;

Аналитическое представление алгоритма имеет следующий вид:

CYK алгоритм = {[u] A1 * B || C * S(v)} *A2

B = C = {[w] B1}

u – условие: Обработаны все диагонали;

v – условие: Все ветви обработали выделенные им элементы;

w – условие: Вычислены все элементы, выделенные для обработки;

A1 – оператор: Распределить количество элементов, обрабатываемой диагонали, по n потокам;

B, C – оператор: Запустить ветвь (i);

A2 – оператор: Проверить наличие аксиомы в элементе R[n, 1];

B1 – оператор: Вычислить элемент.

Граф-схема CYK алгоритма синхронного вычисления элементов диагоналей показана на рис. 4 и ЗАПУСТИТЬ ВЕТВЬ(i) на рис. 5.

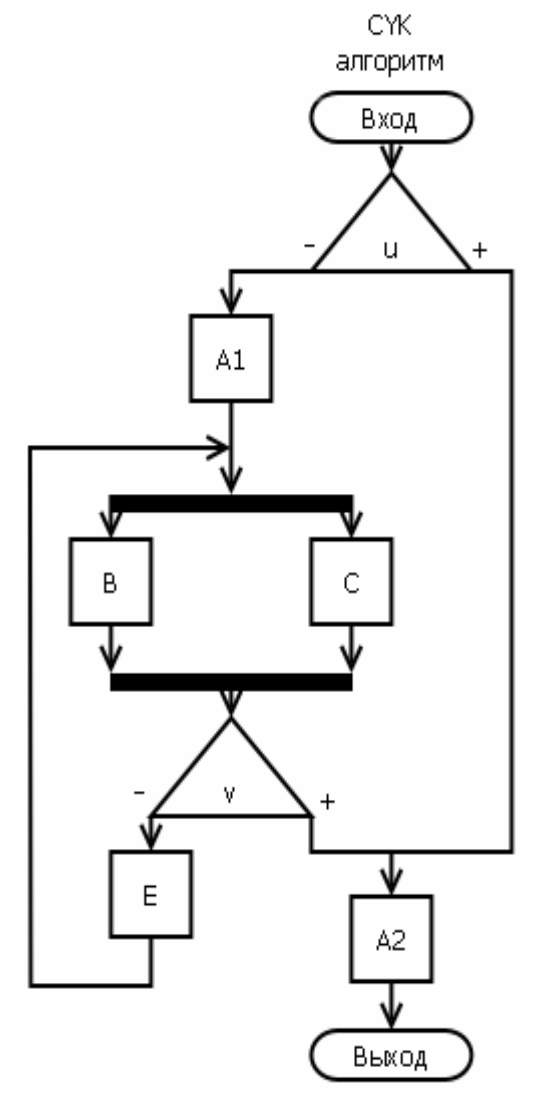

Рис. 4

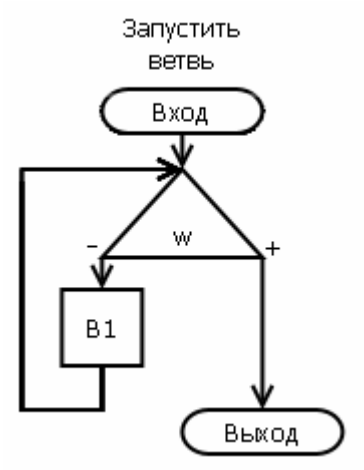

Рис. 5

Использование в инструментарии всех трех форм представления дает более полную картину о конструируемом алгоритме, чем любое из них по отдельности. Соответственно, что изменение любого из этих представлений в инструментарии автоматически приведет к соответствующему преобразованию остальных.

2. Грамматики структурного проектирования

В основу проектирования алгоритмов в алгебре алгоритмики положена концепция матричных грамматик структурного проектирования, которые совмещают аппарат САА-М с теорией языковых мультипроцессоров. Грамматики структурного проектирования (ГСП) [16, 17] были ориентированы на решение важных проблем синтаксиса (описание синтаксически правильных цепочек), семантики (описание не только синтаксически, но и семантически правильных цепочек языка) и прагматики (практическое применение аппарата ГСП для разработки ряда конкретного программного обеспечения).

Под ГСП будем понимать некую формальную систему G = (T, N, R, P, U), составляющими которой являются следующие компоненты:

- T – терминальный алфавит;
- N – нетерминальный алфавит, которому принадлежат метапеременные логические, операторные, объектные, АТД, АТП, характеризующие разные уровни проектирования;

• U – механизм управления выводом;

• R – принадлежит нетерминальному алфавиту, это аксиома которая идентифицирует проектированный класс;

• P – совокупность помеченных постановок логического, операторного, объектного типов, детализирующие АТП, АТД и используются при проектировании алгоритмов и программ.

Рассматриваемые терминальный алфавит, состоящий из совокупности базисных условий, операторов, объектов, абстрактный тип данных (АТД), абстрактный тип памяти (АТП), определяющие степень конкретизации проектированного класса алгоритмов и программ. А так же совокупность разделителей – символов операций сигнатуры САА-М, скобок, ограничителей и др.

Проектируемый класс программ L(G) = {X | R => X} подмножество F(T), которые выводятся из аксиомы в ГСП, создает ассоциированный с классом язык, который порожден данной грамматикой. С помощью механизма U последовательного, параллельного или комбинированного управления выводом в ГСП, реализуются контекстные зависимости по памяти и данным между проектируемыми программными модулями. Следует заметить, что аппарат ГСП положено в основу метода МСПП и его инструментария – системы МУЛЬТИПРОЦЕССИСТ.

В развиваемом инструментарии в основу решения проблемы проектирования сложных программ, положены матричные ГСП (включая и метаправила их конструирования). С последовательным, параллельным или комбинированным применением подстановок в обобщенных матричных продукциях. При этом подстановки, которые применяются последовательно, записываются в строку и разделяются точкой с запятой, а параллельно – в столбец.

Пример 2. Проиллюстрируем процесс взаимосвязанного проектирования структур управления, памяти и данных на примере вышеописанного CYK алгоритма синхронного вычисления элементов диагоналей с применением основы матричных ГСП. А именно П-программы которая порождает разные классы многоуровневых алгоритмов и программ [16, 17]. Суть многоуровневой обработки состоит в распределении обрабатываемой диагонали на попарно непересекающиеся подмассивы для одновременной обработки по параллельным ветвям Ai, и в дальнейшем получениями промежуточных результатов. При этом воспользуемся параметрическим описанием продукций, который обеспечивает саморедактирование ГСП, поданной в схематической форме: наращивание параллельных ветвей и ассоциированных с ними переменных MACi и Ф, выполняется с помощью изменения параметра i от i=1 до $i = n$. Схема ГСП П-Программы состоит из таких обобщенных матричных продукций:

m_0: ‖ R –> ПРС(MAC) x СБОРКА(Ф) ‖

m_1: ‖ MAC –> MAC_1, MAC
ПРС –> $A_1(t)$ v ПРС; t –> MAC_1
Ф –> $Ф_1$, Ф ‖

m_2: ‖ MAC –> MAC_{i+1}, MAC
$A_i(MAC_i)$ v ПРС –>$A_i(MAC_i)$ v $A_{i+1}(t)$ v ПРС
t –> MAC_{i+1}
Ф –> $Ф_{i+1}$, Ф ‖

m_3: ‖ MAC –> MAC_{i+1}
$A_i(MAC_i)$ v ПРС –>$A_i(MAC_i)$ v $A_{i+1}(t)$
t –> MAC_{i+1}
Ф –> $Ф_{i+1}$ ‖

Подключая к продукциям ГСП П-программы матричные продукции m3, m4 получаем ГСП порождающую алгоритм синхронного вычисления элементов диагоналей:

m_3: ‖ Ai(MACi)-> Вычислить элемент(MACi),
MACi->Mi,Ф->Li ‖

m_4: ‖ СБОРКА(L1,..,Lk)->Проверить наличие
аксиомы в элементе(L1,..,Lk) ‖

где, «Вычислить элемент» и «Проверить наличие аксиомы в элементе» – вышеспроектированы в разд. 1 *Mi* – часть диагонали обрабатываемая за *Ai* веткой, *Li* – обработанная часть диагонали за *Ai* ветвью. «Проверить наличие аксиомы в элементе (*L1*,..,*Lk*)» – поиск аксиомы.

В результате, имея вначале последовательный алгоритм синтаксического анализа со сложностью $n^{2,83}$ [15], получаем

параллельный алгоритм синхронного вычисления элементов диагоналей с линейной сложностью 2(n–1) [14] на модели PRAM [5].

3. Средства проектирования и синтеза параллельных объектно-ориентированных программ

Использование аппарата алгебры при разработке программного обеспечения является богатым источником для исследования, благодаря его мощным математическим основам, мощным средствам для использования абстракций программ и программных преобразований. В проектировании параллельного программного обеспечения алгебраическая техника часто и прежде всего, используется для соответствующей декомпозиции предметной области, с целью получения необходимых алгоритмических структур зависимостей в этой области. Для накопления и использования знаний о предметной области, возможности манипулирования проблемно ориентированными объектами и автоматизации генерации программного кода используется алгебра алгоритмов [1–4], где детально разрабатываются полезные проблемно ориентированные наборы действий, условий и абстракций для поддержки этапов жизненного цикла проектирования программного обеспечения.

Главная идея использования алгебры алгоритмов для проектирования программ заключается в пошаговом описании схем алгоритмов сверху вниз, детализируя операции в терминах САА-М [3, 4].

На каждом шаге проектирования, система в диалоге с пользователем позволяет ему выбирать только те действия, выполнение которых в дереве алгоритма не нарушает синтаксической корректности схемы. После того, как алгоритм разработан, диалоговый конструктор может выполнять синтез программы. Это осуществляется путем реализации элементарных операторов и условий на целевом языке программирования, а также других программных фрагментов в дереве алгоритма. В процессе синтеза управляющие конструкции схемы отображаются в соответствующие операторы языка программирования, а вместо базисных элементов подставляются их реализации на этом же языке. На вход синтезатора поступает информация из базы данных алгебраических описаний, содержащий каркасное описание основного класса приложения (без реализаций методов), в который выполняется подстановка синтезированного кода [18, 19].

База данных алгебраических описаний инструментария разработчика содержит следующие компоненты:

1) стратегии обработки (регулярные схемы, которые представляют классы алгоритмов);

2) базовые предикаты и операторы, представленные в трех формах (алгебраической, естественно-лингвистической и графовой) и их реализация на необходимом целевом языке программирования;

3) метаправила для проектирования схем алгоритмов;

4) прикладные алгоритмы;

5) распределенные гиперсхемы.

Базовые инструментальные средства разработчика обеспечат основу для визуального проектирования граф-схем алгоритмов путем интерактивного конструирования синтаксически правильных программ. В разрабатываемом инструментарии будут реализованы средства параметрического управления генерацией САА-схем на основе схем высшего уровня (гиперсхем) [7, 11, 20].

4. Архитектура онлайнового инструментального средства проектирования параллельных и распределенных алгоритмов и программ

Предлагаемый инструментарий состоит из компонентов, показанных на рис. 6.

Как упоминалось ранее, с помощью инструментария осуществляется диалоговое проектирование и синтез объектно-ориентированных параллельных алгоритмов и программ с использованием элементов базы знаний. А так же их последующая отладка, эксплуатация, реинженерия и т.д.

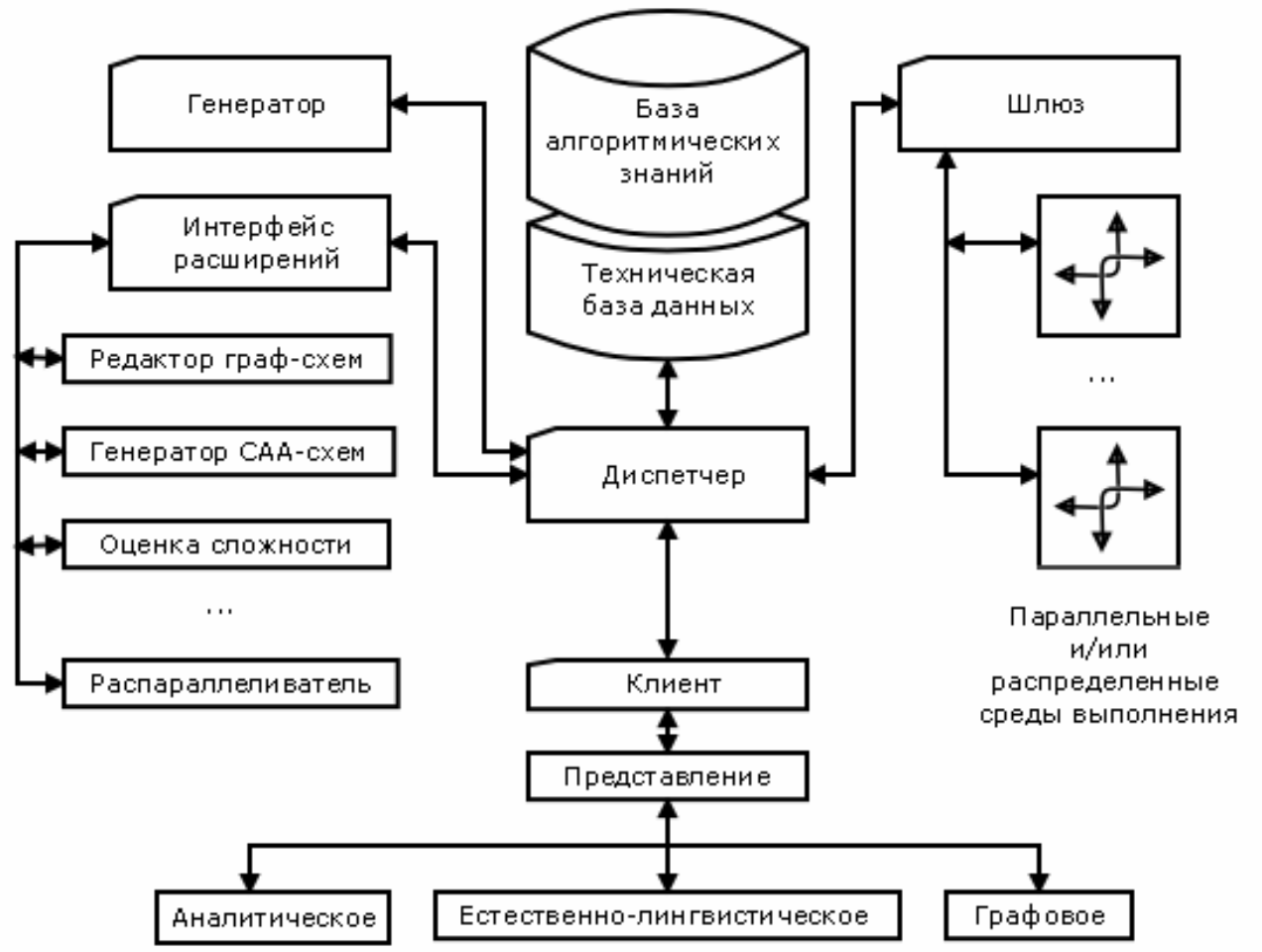

Рис. 6

Отметим, что компоненты инструментария автономны [21], гибко связанны и имеют согласованный протокол обмена данными, т. е. данная модель, по сути, является сервисно-ориентированной [22].

Клиент представляет собой интерфейс для диалогового взаимодействия пользователя с инструментарием и его ресурсами. А именно возможность проектирования последовательного или параллельного алгоритма в вышеупомянутых формах и синтеза исходного кода в целевом языке программирования, а так же его дальнейшей эксплуатации (запуск в доступных современных вычислительных средах, распараллеливание и другое).

Диспетчер – ядро инструментария, организовывающее связь между базой данных, генератором, расширениями, с помощью шлюза, с современными средами выполнения и пользователями (посредством клиентов). Организовывает проектирование, синтез и выполнение (отладка, реинженерия) параллельных программ в современных средах выполнения.

База данных состоит из: базы алгоритмических знаний инструментария и технической базы данных.

База алгоритмических знаний инструментария вмещает следующие разделы:

• схемы алгоритмов (разработанные алгоритмы из разных предметных областей);

• стратегии обработки (схемы, которые описывают классы алгоритмов и подлежат дальнейшей детализации);

• метаправила свертки, развертки и трансформации (обеспечивают абстрагирование, детализацию и переинтерпретацию схем, а также содержат тождества и соотношения для преобразования схем);

• базисные понятия и их программные реализации (ориентированные на проектирование алгоритмов и синтез программ в данной предметной области на избранном целевом языке);

• графические элементы, используемые для представления граф-схем алгоритмов (различные виды стрелок, блоков и др.).

Техническая база данных содержит настройки и данные пользователей инструментария, о зарегистрированных сред выполнения, расширений, а также настройки и стратегии работы инструментария в целом.

Генератор – это сервис, осуществляющий синтез программ. А именно, по дереву, полученному в процессе конструирования алгоритмов, реализациями элементарных операторов и условий целевого объектно-ориентированного языка программирования и другими фрагментами.

Интерфейс расширений – это сервис, ориентированный на расширение возможностей инструментария путем включения дополнительных сервисов для работы с алгоритмами и программами. На пример, оценка сложности алгоритма [23].

Редактор граф-схем ориентирован на визуальное представление алгоритмов. При этом изменения, внесенные во время редактирования, соответствующим образом отобразятся на другие представления алгоритма.

Параметрически управляемый генератор САА-схем [24] ориентирован на синтез схем алгоритмов и программ. Посредством спецификаций более высокого уровня, называемых регулярными гиперсхемами (РГС) [11, 20]. РГС применяются, в частности, для представления алгоритмов управления выводом в грамматиках структурного проектирования (ГСП) [16, 17]. Проектирование гиперсхем, как и САА-схем, выполняется в диалоговом режиме.

Распараллеливатель – это сервис, ориентирован на реинженерию последовательных алгоритмов и программ с целью получения их параллельных версий.

Шлюз – это сервис, обеспечивающий запуск, анализ и получение результатов выполнения программ в современных средах выполнения.

Заключение

В данной работе предлагается концепция развития инструментария, ориентированного на разработку параллельных алгоритмов и программ для современных параллельных сред выполнения. В основу проектирования алгоритмов в этом инструментарии положен аппарат алгебры алгоритмики, а именно концепция грамматик структурного проектирования, которая совмещает модифицированные системы алгоритмических алгебр с теорией языковых мультипроцессоров.

Языковые мультипроцессоры спроектированы в форме грамматик структурного проектирования. В частности для метода Кока – Янгера – Касами использованы матричные ГСП. Спроектирована архитектура онлайнового инструментального средства проектирования параллельных и распределенных алгоритмов.

1. *Gluschkow, W.M., Zeitlin, G.E., Justchenko, J.L.* Algebra. Sprachen. Programmierung. Akademie-Verlag, Berlin, 1980. – 340 p.
2. *Цейтлин Г.Е.* Введение в алгоритмику. Киев: Сфера. – 1998. – 310 с.
3. *Doroshenko A., Tseitlin G.* Models and Parallel Programming Abstractions to Enhance Concurrency of Parallel Programs, Fundamenta Informaticae. – 2004. – Vol. 60, N. 1-4. – P. 99–111.
4. *Андон Ф.И., Дорошенко А.Е., Цейтлин Г.Е., Яценко Е.А.* Алгеброалгоритмические модели и методы параллельного программирования. – Киев: Академпериодика, 2007. – 634 с.
5. *Дорошенко А.Е.*. Математические модели и методы организации высокопроизводительных параллельных вычислений. Алгебродинамический подход. – Киев: Наук. думка, 2000. – 177 с.
6. *Цейтлин Г.Е., Яценко Е.А.* Элементы алгебраической алгоритмики и объектно-ориентированный синтез параллельных программ // Математические машины и системы. – 2003. – № 2. – С. 64–76.
7. *Яценко Е.А.* Алгебры гиперсхем и интегрированный инструментарий синтеза программ в современных объектно-ориентированных средах // Кибернетика и системный анализ. – 2004. – № 1. – С. 47 – 52.

8. *Яценко О.А.* Середовище конструювання алгоритмічних знань та інструментарій синтезу програм // Проблеми програмування. — 2006. — № 2-3. — С. 349–359.
9. *Doroshenko, A.E., Shevchenko, R. A* Rewriting Framework for Rule-Based Programming Dynamic Applications // Fundamenta Informaticae. – 2006. – 72. – P. 95–108.
10. *TermWar* – http://www.gradsoft.com.ua/products/termware_rus.html
11. *Ющенко Е.Л., Цейтлин Г.Е., Грицай В.П., Терзян Т.К.* Многоуровневое структурное проектирование программ: Теоретические основы, инструментарий. – М.: Финансы и статистика, 1989. – 208 с.
12. *Калужнин Л.А.* Об алгоритмизации математических задач // Проблемы кибернетики. – 1959. – Вып. 2. – С. 51–69.
13. *Ющенко Е.Л., Цейтлин Г.Е., Галушка А.В.* Алгебро-грамматические спецификации и синтез структурированных схем программ // Кибернетика. – 1989. – № 6. – С. 5–16.
14. *Глушков В.М., Цейтлин Г.Е., Ющенко Е.Л.* Методы символьной мультиобработки. – Киев: Наук. думка, 1980. – 252 с.
15. *Янгер Д.Х.* Распознавание и анализ контекстно-свободных языков за время n3 // Проблемы математической логики. – М.: Мир, 1970. – С. 344–362.
16. *Цейтлин Г.Е., Иовчев В.А., Мусихин А.А.* Ментальные аспекты методов символьной мультиобработки // Проблеми програмування. – 2008. – № 1. – Р. 60–67.
17. *Цейтлин Г.Е., Суржко С.В., Ющенко К.Л., Шевченко А.И.* Алгоритмические алгебры. – Киев, 1997. – 342 с.
18. *Яценко Е.А., Мохница А.С.* Инструментальные средства конструирования синтаксически правильных параллельных алгоритмов и программ // Проблемы программирования. – 2004. – № 2-3. – С. 444 – 450.
19. *Цейтлин Г.Е., Яценко Е.А.* Элементы алгебраической алгоритмики и объектно-ориентированный синтез параллельных программ // Математические машины и системы. – 2003. – № 2. – С. 64 – 76.
20. *Цейтлин Г.Е.* Алгебры Глушкова и теория клонов // Кибернетика и системный анализ. – 2003. – № 4. – С. 48–58.
21. *IBM Autonomic Computing*. — http://www.research.ibm.com/autonomic/
22. *Дорошенко А.Е., Алистратов О.В., Тырчак Ю.М., Розенблат А.П.* Системы GRID-вычислений — перспектива для научных исследований // Проблеми програмування. – 2005. – № 1. – Р. 14–38.
23. *Дорошенко А.Е., Жереб К.А., Яценко Е.А.* Об оценке сложности и координации вычислений в многопоточных программах // Проблеми програмування. – 2007. – № 2. – С. 41–55.
24. *Яценко Е.А.* Алгебры гиперсхем и интегрированный инструментарий синтеза программ в современных объектно-ориентированных средах. // Кибернетика и системный анализ. – 2004. – № 1. – С. 47–52.

Получено 19.06.2009

Об авторах:

Дорошенко Анатолий Ефимович,
доктор физико-математических наук,
профессор, заведующий отделом теории компьютерных вычислений,

Цейтлин Георгий Евсеевич,
доктор физико-математических наук,
профессор, ведущий научный сотрудник,

Иовчев Владимир Александрович,
аспирант Института программных систем НАН Украины.

Место работы авторов:

Институт программных систем
НАН Украины,
03680, Киев-187,
Проспект Академика Глушкова, 40.
e-mail:
dor@isofts.kiev.ua,
tseytlin@vikno.relc.com,
iovchev.v@gmail.com